BOOK 8: EVENTS
Chapter 1: Event Basics

WHAT ARE EVENTS

Events are things that happen on a web page:

- User clicks a button
- User types in an input
- Mouse moves over an element
- Page finishes loading
- Form is submitted
- Key is pressed

JavaScript can listen for these events and respond to them.

ADDING EVENT LISTENERS

The modern way to handle events:

```
let button = document.getElementById("myButton")

button.addEventListener("click", function() {
 console.log("Button was clicked!")
})
```

addEventListener takes two arguments:
1. The event name (click, mouseover, keydown, etc.)
2. A function to run when the event happens

The function is called an event handler or callback.

ONCLICK PROPERTY

Older way to handle clicks:

```
let button = document.getElementById("myButton")

button.onclick = function() {
 console.log("Button was clicked!")
}
```

This works but has a limitation:

```
button.onclick = function() {
 console.log("First handler")
}

button.onclick = function() {
 console.log("Second handler")
}
```

Only "Second handler" runs. The first was overwritten.

addEventListener does not have this problem:

```
button.addEventListener("click", function() {
 console.log("First handler")
})

button.addEventListener("click", function() {
 console.log("Second handler")
})
```

Both handlers run.

INLINE EVENT HANDLERS

HTML:

<button onclick="handleClick()">Click me</button>

JavaScript:

```
function handleClick() {
 console.log("Clicked!")
}
```

This is the oldest way. Avoid it. It mixes HTML and JavaScript.

REMOVING EVENT LISTENERS

To remove a listener, you need a reference to the function:

```
function handleClick() {
 console.log("Clicked!")
}

button.addEventListener("click", handleClick)

button.removeEventListener("click", handleClick)
```

Anonymous functions cannot be removed:

```
button.addEventListener("click", function() {
 console.log("This cannot be removed")
})
```

THE EVENT OBJECT

When an event fires, JavaScript passes an event object:

```
button.addEventListener("click", function(event) {
```

```
 console.log(event)
 console.log(event.type)
 console.log(event.target)
})
```

The event object contains information about what happened.

Common properties:
event.type - the event name (click, keydown, etc.)
event.target - the element that triggered the event
event.currentTarget - the element the handler is attached to
event.timeStamp - when the event occurred

USING THE EVENT TARGET

Get information about what was clicked:

```
button.addEventListener("click", function(event) {
 console.log(event.target.id)
 console.log(event.target.textContent)
 console.log(event.target.className)
})
```

Change the clicked element:

```
button.addEventListener("click", function(event) {
 event.target.textContent = "Clicked!"
 event.target.style.backgroundColor = "green"
})
```

COMMON EVENT TYPES

Mouse events:
click - left mouse button click

dblclick - double click
mousedown - mouse button pressed
mouseup - mouse button released
mousemove - mouse moves
mouseenter - mouse enters element
mouseleave - mouse leaves element
contextmenu - right click

Keyboard events:
keydown - key pressed down
keyup - key released
keypress - key pressed (deprecated)

Form events:
submit - form submitted
change - input value changed
input - input value changing
focus - element focused
blur - element lost focus

Window events:
load - page finished loading
resize - window size changed
scroll - page scrolled

Touch events:
touchstart - finger touches screen
touchmove - finger moves on screen
touchend - finger leaves screen

MULTIPLE EVENTS ON ONE ELEMENT

```
let input = document.getElementById("name")

input.addEventListener("focus", function() {
 console.log("Input focused")
})
```

```
input.addEventListener("blur", function() {
 console.log("Input blurred")
})

input.addEventListener("input", function() {
 console.log("Input changed:", this.value)
})
```

ONE HANDLER FOR MULTIPLE ELEMENTS

```
let buttons = document.querySelectorAll("button")

buttons.forEach(button => {
 button.addEventListener("click", function(event) {
 console.log("Clicked:", event.target.textContent)
 })
})
```

PRACTICAL EXAMPLES

Toggle button text:

```
let button = document.getElementById("toggle")
let isOn = false

button.addEventListener("click", function() {
 isOn = !isOn
 this.textContent = isOn ? "ON" : "OFF"
})
```

Count clicks:

```
let count = 0
```

```
let display = document.getElementById("count")
let button = document.getElementById("counter")

button.addEventListener("click", function() {
 count++
 display.textContent = count
})
```

Show character count:

```
let textarea = document.getElementById("message")
let charCount = document.getElementById("char-count")

textarea.addEventListener("input", function() {
 charCount.textContent = this.value.length + " characters"
})
```

PRACTICE

Add a click event listener to a button.

Access the event object and log its properties.

Add multiple event types to one element.

Add the same handler to multiple elements.

Create a counter that increments on click.

WHAT YOU LEARNED

You learned events are things that happen on a page.
You learned addEventListener attaches handlers.
You learned the event object contains event information.

You learned event.target is the element that triggered the event.
You learned common event types.
You learned to attach handlers to multiple elements.

The next chapter teaches you about event flow and propagation.

BOOK 8: EVENTS

Chapter 2: Event Flow

EVENT PROPAGATION

When you click an element, you also click its parents:

HTML:

```
<div id="outer">
 <div id="inner">
 <button id="button">Click</button>
 </div>
</div>
```

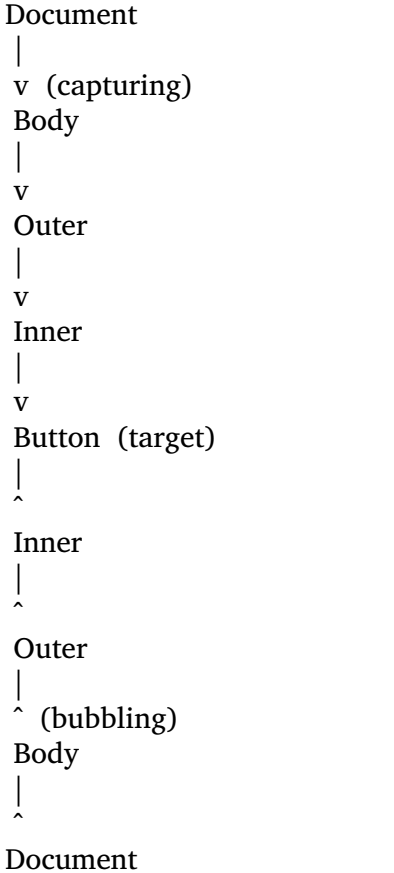
Click the button and you click:
- The button
- The inner div
- The outer div
- The body
- The document

Events propagate through the DOM tree.

THREE PHASES

Event propagation has three phases:

1. Capturing phase - event travels down from document to target
2. Target phase - event reaches the target element
3. Bubbling phase - event travels up from target to document



EVENT BUBBLING

By default, events bubble up:

```
let outer = document.getElementById("outer")
let inner = document.getElementById("inner")
let button = document.getElementById("button")
```

```
outer.addEventListener("click", function() {
 console.log("Outer clicked")
})

inner.addEventListener("click", function() {
 console.log("Inner clicked")
})

button.addEventListener("click", function() {
 console.log("Button clicked")
})
```

Click the button and you see:

```
Button clicked
Inner clicked
Outer clicked
```

The event bubbles up from button to inner to outer.

EVENT CAPTURING

Listen during the capturing phase by setting third argument to true:

```
outer.addEventListener("click", function() {
 console.log("Outer clicked")
}, true)

inner.addEventListener("click", function() {
 console.log("Inner clicked")
}, true)

button.addEventListener("click", function() {
 console.log("Button clicked")
}, true)
```

Click the button and you see:

Outer clicked
Inner clicked
Button clicked

The event captures down from outer to inner to button.

COMBINING PHASES

```
outer.addEventListener("click", function() {
 console.log("Outer - capturing")
}, true)

outer.addEventListener("click", function() {
 console.log("Outer - bubbling")
}, false)

inner.addEventListener("click", function() {
 console.log("Inner - capturing")
}, true)

inner.addEventListener("click", function() {
 console.log("Inner - bubbling")
}, false)

button.addEventListener("click", function() {
 console.log("Button clicked")
})
```

Click the button:

Outer - capturing
Inner - capturing
Button clicked
Inner - bubbling
Outer - bubbling

STOPPROPAGATION

Prevent the event from continuing:

```
button.addEventListener("click", function(event) {
 console.log("Button clicked")
 event.stopPropagation()
})

inner.addEventListener("click", function() {
 console.log("Inner clicked")
})

outer.addEventListener("click", function() {
 console.log("Outer clicked")
})
```

Click the button:

Button clicked

The event stops at the button. Inner and outer handlers do not run.

STOPIMMEDIATEPROPAGATION

Stop all handlers on current element and prevent propagation:

```
button.addEventListener("click", function(event) {
 console.log("First handler")
 event.stopImmediatePropagation()
})

button.addEventListener("click", function() {
```

```
 console.log("Second handler")
})

outer.addEventListener("click", function() {
 console.log("Outer clicked")
})
```

Click the button:

First handler

Both the second handler and parent handlers are stopped.

EVENT.TARGET VS EVENT.CURRENTTARGET

event.target - the element that triggered the event
event.currentTarget - the element the handler is attached to

HTML:

```
<div id="parent">
 <button>Button 1</button>
 <button>Button 2</button>
</div>
```

JavaScript:

```
let parent = document.getElementById("parent")

parent.addEventListener("click", function(event) {
 console.log("target:", event.target.textContent)
 console.log("currentTarget:", event.currentTarget.id)
})
```

Click Button 1:

target: Button 1

currentTarget: parent

The target is the button. The currentTarget is the parent (where the handler is attached).

PREVENTDEFAULT

Prevent the default browser behavior:

```
let link = document.querySelector("a")

link.addEventListener("click", function(event) {
 event.preventDefault()
 console.log("Link click prevented")
})

let form = document.querySelector("form")

form.addEventListener("submit", function(event) {
 event.preventDefault()
 console.log("Form submit prevented")
})

let checkbox = document.querySelector("input[type='checkbox']")

checkbox.addEventListener("click", function(event) {
 event.preventDefault()
 console.log("Checkbox toggle prevented")
})
```

Use when you want to handle the action yourself.

CHECKING IF PREVENTABLE

```
let link = document.querySelector("a")

link.addEventListener("click", function(event) {
 if (event.cancelable) {
 event.preventDefault()
 }
})
```

PRACTICAL EXAMPLES

Close modal when clicking outside:

```
let modal = document.getElementById("modal")
let modalContent = document.querySelector(".modal-content")

modal.addEventListener("click", function(event) {
 if (event.target = = = modal) {
 modal.classList.remove("active")
 }
})

modalContent.addEventListener("click", function(event) {
 event.stopPropagation()
})
```

Dropdown that closes on outside click:

```
let dropdown = document.getElementById("dropdown")

document.addEventListener("click", function(event) {
 if (!dropdown.contains(event.target)) {
 dropdown.classList.remove("open")
 }
})

dropdown.addEventListener("click", function(event) {
 event.stopPropagation()
```

```
 this.classList.toggle("open")
})
```

Confirm before leaving:

```
let form = document.getElementById("form")
let hasChanges = false

form.addEventListener("input", function() {
 hasChanges = true
})

window.addEventListener("beforeunload", function(event) {
 if (hasChanges) {
 event.preventDefault()
 event.returnValue = ""
 }
})
```

PRACTICE

Log events during bubbling phase.

Log events during capturing phase.

Use stopPropagation to stop bubbling.

Use preventDefault to stop default behavior.

Build a component that uses event flow.

WHAT YOU LEARNED

You learned events propagate through the DOM.

You learned the three phases: capturing, target, bubbling.
You learned events bubble up by default.
You learned addEventListener third argument enables capturing.
You learned stopPropagation stops event flow.
You learned preventDefault stops default browser behavior.
You learned the difference between target and currentTarget.

The next chapter teaches you about event delegation.

BOOK 8: EVENTS

Chapter 3: Event Delegation

THE PROBLEM

Adding listeners to many elements:

HTML:

```
<ul id="list">
 <li>Item 1</li>
 <li>Item 2</li>
 <li>Item 3</li>
 <li>Item 4</li>
 <li>Item 5</li>
</ul>
```

Bad approach:

```
let items = document.querySelectorAll("li")

items.forEach(item => {
 item.addEventListener("click", function() {
 console.log(this.textContent)
 })
})
```

Problems:

- 5 event listeners for 5 items
- New items added later will not have listeners
- Memory usage increases with more items

EVENT DELEGATION SOLUTION

Attach one listener to the parent:

```
let list = document.getElementById("list")

list.addEventListener("click", function(event) {
 if (event.target.tagName === "LI") {
 console.log(event.target.textContent)
 }
})
```

Benefits:
- One event listener instead of many
- Works for items added later
- Less memory usage
- Cleaner code

HOW IT WORKS

When you click an li, the event bubbles up to the ul.

The handler on the ul fires.

event.target tells you which li was clicked.

You check if the target is an li and handle it.

CHECKING THE TARGET

By tag name:

```
list.addEventListener("click", function(event) {
 if (event.target.tagName === "LI") {
 console.log("Li clicked")
 }
})
```

By class:

```
list.addEventListener("click", function(event) {
 if (event.target.classList.contains("item")) {
 console.log("Item clicked")
 }
})
```

By matches:

```
list.addEventListener("click", function(event) {
 if (event.target.matches("li.item")) {
 console.log("Li with item class clicked")
 }
})
```

HANDLING NESTED ELEMENTS

HTML:

```
<ul id="list">
 <li>
 <span class="text">Item 1</span>
 <button class="delete">X</button>
 </li>
 <li>
```

```
 <span class="text">Item 2</span>
 <button class="delete">X</button>
 </li>
</ul>
```

Problem:

```
list.addEventListener("click", function(event) {
 if (event.target.tagName === "LI") {
 console.log("Li clicked")
 }
})
```

Clicking the span or button, target is not li.

Solution with closest:

```
list.addEventListener("click", function(event) {
 let li = event.target.closest("li")

 if (li) {
 console.log("Li clicked:", li.textContent)
 }
})
```

closest finds the nearest ancestor matching the selector (including the element itself).

HANDLING DIFFERENT ACTIONS

HTML:

```
<ul id="todo-list">
 <li data-id="1">
 <span>Task 1</span>
 <button class="edit">Edit</button>
 <button class="delete">Delete</button>
```

```
 </li>
 <li data-id="2">
 <span>Task 2</span>
 <button class="edit">Edit</button>
 <button class="delete">Delete</button>
 </li>
</ul>
```

JavaScript:

```
let list = document.getElementById("todo-list")

list.addEventListener("click", function(event) {
 let li = event.target.closest("li")
 if (!li) return

 let id = li.dataset.id

 if (event.target.classList.contains("delete")) {
 deleteItem(id)
 } else if (event.target.classList.contains("edit")) {
 editItem(id)
 }
})

function deleteItem(id) {
 console.log("Delete item", id)
 let li = document.querySelector(`li[data-id="${id}"]`)
 li.remove()
}

function editItem(id) {
 console.log("Edit item", id)
}
```

DYNAMIC CONTENT

Event delegation handles dynamically added elements:

```
let list = document.getElementById("list")
let addBtn = document.getElementById("add")
let count = 0

list.addEventListener("click", function(event) {
 if (event.target.tagName === "LI") {
 console.log("Clicked:", event.target.textContent)
 }
})

addBtn.addEventListener("click", function() {
 count++
 let li = document.createElement("li")
 li.textContent = `New Item ${count}`
 list.appendChild(li)
})
```

New items work automatically. No need to add listeners to them.

DELEGATION WITH FORMS

HTML:

```
<form id="settings">
 <label>
 <input type="checkbox" name="notifications"> Notifications
 </label>
 <label>
 <input type="checkbox" name="darkMode"> Dark Mode
 </label>
 <label>
 <input type="checkbox" name="sound"> Sound
 </label>
</form>
```

JavaScript:

```
let form = document.getElementById("settings")

form.addEventListener("change", function(event) {
 if (event.target.type === "checkbox") {
 let setting = event.target.name
 let value = event.target.checked

 console.log(`${setting}: ${value}`)
 saveSetting(setting, value)
 }
})

function saveSetting(name, value) {
 localStorage.setItem(name, value)
}
```

DELEGATION PATTERN

General pattern for event delegation:

```
function delegate(parent, selector, eventType, handler) {
 parent.addEventListener(eventType, function(event) {
 let target = event.target.closest(selector)

 if (target && parent.contains(target)) {
 handler.call(target, event)
 }
 })
}

let list = document.getElementById("list")

delegate(list, "li", "click", function(event) {
 console.log("Clicked:", this.textContent)
})
```

```
delegate(list, ".delete", "click", function(event) {
 this.closest("li").remove()
})
```

PRACTICAL EXAMPLES

Navigation menu:

```
let nav = document.getElementById("nav")

nav.addEventListener("click", function(event) {
 let link = event.target.closest("a")

 if (link && nav.contains(link)) {
 event.preventDefault()

 document.querySelectorAll("nav a").forEach(a =>
a.classList.remove("active"))
 link.classList.add("active")

 let page = link.dataset.page
 loadPage(page)
 }
})
```

Table row selection:

```
let table = document.getElementById("data-table")

table.addEventListener("click", function(event) {
 let row = event.target.closest("tr")

 if (row && row.parentElement.tagName === "TBODY") {
 document.querySelectorAll("tr.selected").forEach(r =>
r.classList.remove("selected"))
 row.classList.toggle("selected")
```

```
 }
})
```

Accordion:

```
let accordion = document.getElementById("accordion")

accordion.addEventListener("click", function(event) {
 let header = event.target.closest(".accordion-header")

 if (header) {
 let item = header.closest(".accordion-item")
 let isOpen = item.classList.contains("open")

 document.querySelectorAll(".accordion-item").forEach(i => {
 i.classList.remove("open")
 })

 if (!isOpen) {
 item.classList.add("open")
 }
 }
})
```

WHEN TO USE DELEGATION

Use delegation when:
- You have many similar elements
- Elements are added dynamically
- You want cleaner code
- You want better performance

Do not use delegation when:
- You have one or few elements
- Elements need very different handlers
- You need to stop propagation frequently

PRACTICE

Delegate click events on a list.

Handle clicks on nested elements using closest.

Handle multiple action buttons with delegation.

Add new items and verify delegation works.

Create a reusable delegation function.

WHAT YOU LEARNED

You learned delegation attaches one handler to a parent.
You learned event.target identifies the clicked element.
You learned closest finds the nearest matching ancestor.
You learned delegation works for dynamic content.
You learned when to use delegation.

The next chapter teaches you about mouse events.
BOOK 8: EVENTS
Chapter 4: Mouse Events

CLICK EVENTS

click - fires when mouse button pressed and released:

```
let button = document.getElementById("button")
```

```
button.addEventListener("click", function(event) {
 console.log("Clicked!")
})
```

dblclick - fires on double click:

```
button.addEventListener("dblclick", function(event) {
 console.log("Double clicked!")
})
```

MOUSE BUTTON EVENTS

mousedown - fires when any mouse button pressed:

```
element.addEventListener("mousedown", function(event) {
 console.log("Mouse button down")
})
```

mouseup - fires when mouse button released:

```
element.addEventListener("mouseup", function(event) {
 console.log("Mouse button up")
})
```

The sequence for a click:
1. mousedown
2. mouseup
3. click

WHICH BUTTON WAS PRESSED

The event.button property:

```
element.addEventListener("mousedown", function(event) {
```

```
 switch (event.button) {
 case 0:
 console.log("Left button")
 break
 case 1:
 console.log("Middle button")
 break
 case 2:
 console.log("Right button")
 break
 }
})
```

contextmenu - fires on right click:

```
element.addEventListener("contextmenu", function(event) {
 event.preventDefault()
 console.log("Right click - showing custom menu")
})
```

MOUSE POSITION

Position relative to viewport:

```
element.addEventListener("click", function(event) {
 console.log("clientX:", event.clientX)
 console.log("clientY:", event.clientY)
})
```

Position relative to document:

```
element.addEventListener("click", function(event) {
 console.log("pageX:", event.pageX)
 console.log("pageY:", event.pageY)
})
```

Position relative to screen:

```
element.addEventListener("click", function(event) {
 console.log("screenX:", event.screenX)
 console.log("screenY:", event.screenY)
})
```

Position relative to element:

```
element.addEventListener("click", function(event) {
 console.log("offsetX:", event.offsetX)
 console.log("offsetY:", event.offsetY)
})
```

MOUSE MOVEMENT

mousemove - fires when mouse moves:

```
document.addEventListener("mousemove", function(event) {
 console.log(`Mouse at ${event.clientX}, ${event.clientY}`)
})
```

Track mouse position:

```
let display = document.getElementById("coords")

document.addEventListener("mousemove", function(event) {
 display.textContent = `X: ${event.clientX}, Y: ${event.clientY}`
})
```

ENTERING AND LEAVING

mouseenter - fires when mouse enters element:

```
element.addEventListener("mouseenter", function() {
```

```
 this.style.backgroundColor = "yellow"
})
```

mouseleave - fires when mouse leaves element:

```
element.addEventListener("mouseleave", function() {
 this.style.backgroundColor = ""
})
```

mouseover and mouseout also exist but bubble and fire for child elements too.

Difference:
mouseenter/mouseleave - do not bubble, ignore children
mouseover/mouseout - bubble, fire for children too

Usually mouseenter/mouseleave are what you want.

HOVER EFFECT

```
let cards = document.querySelectorAll(".card")

cards.forEach(card => {
 card.addEventListener("mouseenter", function() {
 this.classList.add("hover")
 })

 card.addEventListener("mouseleave", function() {
 this.classList.remove("hover")
 })
})
```

DRAG FUNCTIONALITY

Basic drag implementation:

```
let box = document.getElementById("draggable")
let isDragging = false
let offsetX, offsetY

box.addEventListener("mousedown", function(event) {
 isDragging = true
 offsetX = event.clientX - box.offsetLeft
 offsetY = event.clientY - box.offsetTop
 box.style.cursor = "grabbing"
})

document.addEventListener("mousemove", function(event) {
 if (isDragging) {
 box.style.left = (event.clientX - offsetX) + "px"
 box.style.top = (event.clientY - offsetY) + "px"
 }
})

document.addEventListener("mouseup", function() {
 isDragging = false
 box.style.cursor = "grab"
})
```

CSS:

```
#draggable {
 position: absolute;
 cursor: grab;
}
```

DRAWING ON CANVAS

```
let canvas = document.getElementById("canvas")
let ctx = canvas.getContext("2d")
let isDrawing = false
```

```
canvas.addEventListener("mousedown", function(event) {
 isDrawing = true
 ctx.beginPath()
 ctx.moveTo(event.offsetX, event.offsetY)
})

canvas.addEventListener("mousemove", function(event) {
 if (isDrawing) {
 ctx.lineTo(event.offsetX, event.offsetY)
 ctx.stroke()
 }
})

canvas.addEventListener("mouseup", function() {
 isDrawing = false
})

canvas.addEventListener("mouseleave", function() {
 isDrawing = false
})
```

MODIFIER KEYS

Check if Ctrl, Shift, Alt, or Meta was held:

```
element.addEventListener("click", function(event) {
 if (event.ctrlKey) {
 console.log("Ctrl + Click")
 }

 if (event.shiftKey) {
 console.log("Shift + Click")
 }

 if (event.altKey) {
 console.log("Alt + Click")
```

```
 }

 if (event.metaKey) {
 console.log("Meta + Click")
 }
})
```

Multi-select pattern:

```
let items = document.querySelectorAll(".item")

items.forEach(item => {
 item.addEventListener("click", function(event) {
 if (event.ctrlKey || event.metaKey) {
 this.classList.toggle("selected")
 } else {
 items.forEach(i => i.classList.remove("selected"))
 this.classList.add("selected")
 }
 })
})
```

PRACTICAL EXAMPLES

Tooltip following mouse:

```
let tooltip = document.getElementById("tooltip")

document.addEventListener("mousemove", function(event) {
 tooltip.style.left = (event.clientX + 10) + "px"
 tooltip.style.top = (event.clientY + 10) + "px"
})
```

Custom context menu:

```
let menu = document.getElementById("context-menu")
```

```
document.addEventListener("contextmenu", function(event) {
 event.preventDefault()
 menu.style.display = "block"
 menu.style.left = event.clientX + "px"
 menu.style.top = event.clientY + "px"
})

document.addEventListener("click", function() {
 menu.style.display = "none"
})
```

Image zoom on hover:

```
let images = document.querySelectorAll(".zoomable")

images.forEach(img => {
 img.addEventListener("mouseenter", function() {
 this.style.transform = "scale(1.1)"
 })

 img.addEventListener("mouseleave", function() {
 this.style.transform = "scale(1)"
 })
})
```

Range slider:

```
let slider = document.getElementById("slider")
let track = document.getElementById("track")
let thumb = document.getElementById("thumb")
let isDragging = false

thumb.addEventListener("mousedown", function(event) {
 isDragging = true
 event.preventDefault()
})

document.addEventListener("mousemove", function(event) {
 if (isDragging) {
 let rect = track.getBoundingClientRect()
```

```
 let percent = (event.clientX - rect.left) / rect.width
 percent = Math.max(0, Math.min(1, percent))
 thumb.style.left = (percent * 100) + "%"

 let value = Math.round(percent * 100)
 console.log("Value:", value)
 }
})

document.addEventListener("mouseup", function() {
 isDragging = false
})
```

PRACTICE

Handle click, double click, and right click events.

Track and display mouse position.

Implement hover effects.

Create a draggable element.

Use modifier keys with click.

WHAT YOU LEARNED

You learned click, dblclick, mousedown, mouseup events.
You learned event.button identifies which button.
You learned clientX, clientY for mouse position.
You learned mouseenter, mouseleave for hover.
You learned to implement drag functionality.
You learned to check modifier keys.

The next chapter teaches you about keyboard events.

BOOK 8: EVENTS

Chapter 5: Keyboard Events

KEYBOARD EVENT TYPES

keydown - fires when key is pressed:

```
document.addEventListener("keydown", function(event) {
 console.log("Key pressed:", event.key)
})
```

keyup - fires when key is released:

```
document.addEventListener("keyup", function(event) {
 console.log("Key released:", event.key)
})
```

keypress is deprecated. Use keydown instead.

THE KEY PROPERTY

event.key gives the actual key value:

```
document.addEventListener("keydown", function(event) {
 console.log(event.key)
})
```

Examples:
"a", "b", "c" - letter keys
"1", "2", "3" - number keys
"Enter" - enter key
"Escape" - escape key

"ArrowUp", "ArrowDown", "ArrowLeft", "ArrowRight" - arrow keys
"Backspace" - backspace key
"Tab" - tab key
" " - space key
"Shift", "Control", "Alt", "Meta" - modifier keys

THE CODE PROPERTY

event.code gives the physical key position:

```
document.addEventListener("keydown", function(event) {
 console.log("key:", event.key)
 console.log("code:", event.code)
})
```

Press "a":
key: a
code: KeyA

Press Shift + "a":
key: A
code: KeyA

Use key for the character. Use code for the physical key.

CHECKING SPECIFIC KEYS

```
document.addEventListener("keydown", function(event) {
 if (event.key === "Enter") {
 console.log("Enter pressed")
 }

 if (event.key === "Escape") {
 console.log("Escape pressed")
```

```
 }

 if (event.key === "ArrowUp") {
 console.log("Up arrow pressed")
 }
})
```

MODIFIER KEYS

Check if modifier keys are held:

```
document.addEventListener("keydown", function(event) {
 if (event.ctrlKey) {
 console.log("Ctrl is held")
 }

 if (event.shiftKey) {
 console.log("Shift is held")
 }

 if (event.altKey) {
 console.log("Alt is held")
 }

 if (event.metaKey) {
 console.log("Meta (Cmd/Win) is held")
 }
})
```

KEYBOARD SHORTCUTS

```
document.addEventListener("keydown", function(event) {
 if (event.ctrlKey && event.key === "s") {
 event.preventDefault()
```

```
 console.log("Save shortcut")
 saveDocument()
 }

 if (event.ctrlKey && event.key === "z") {
 event.preventDefault()
 console.log("Undo shortcut")
 undo()
 }

 if (event.ctrlKey && event.shiftKey && event.key === "Z") {
 event.preventDefault()
 console.log("Redo shortcut")
 redo()
 }
})
```

PREVENT DEFAULT

Stop default behavior:

```
let input = document.getElementById("numbers-only")

input.addEventListener("keydown", function(event) {
 if (!/[0-9]/.test(event.key) && event.key !== "Backspace" &&
event.key !== "Tab") {
 event.preventDefault()
 }
})
```

This input only accepts numbers.

INPUT VS KEYDOWN

For text input, prefer the input event:

```
let input = document.getElementById("search")

input.addEventListener("input", function() {
 console.log("Current value:", this.value)
})
```

input event fires after the value changes.
keydown fires before the value changes.

Use input for responding to text changes.
Use keydown for shortcuts and navigation.

KEYBOARD NAVIGATION

Arrow key navigation:

```
let items = document.querySelectorAll(".menu-item")
let currentIndex = 0

function updateSelection() {
 items.forEach((item, index) = > {
 item.classList.toggle("selected", index = = = currentIndex)
 })
}

document.addEventListener("keydown", function(event) {
 if (event.key = = = "ArrowDown") {
 event.preventDefault()
 currentIndex = (currentIndex + 1) % items.length
 updateSelection()
 }

 if (event.key = = = "ArrowUp") {
 event.preventDefault()
 currentIndex = (currentIndex - 1 + items.length) % items.length
```

```
 updateSelection()
 }

 if (event.key === "Enter") {
 items[currentIndex].click()
 }
})

updateSelection()
```

KEY REPEAT

When holding a key, keydown fires repeatedly:

```
document.addEventListener("keydown", function(event) {
 if (event.repeat) {
 console.log("Key is being held")
 } else {
 console.log("Key just pressed")
 }
})
```

Ignore repeats:

```
document.addEventListener("keydown", function(event) {
 if (event.repeat) return

 console.log("Key pressed once:", event.key)
})
```

FOCUS AND KEYBOARD EVENTS

Keyboard events fire on the focused element:

```
let input = document.getElementById("name")

input.addEventListener("keydown", function(event) {
 console.log("Key in input:", event.key)
})
```

If no element is focused, events fire on document.

PRACTICAL EXAMPLES

Modal close on Escape:

```
let modal = document.getElementById("modal")

document.addEventListener("keydown", function(event) {
 if (event.key === "Escape" && modal.classList.contains("open"))
{
 modal.classList.remove("open")
 }
})
```

Game controls:

```
let player = { x: 0, y: 0 }
let keys = {}

document.addEventListener("keydown", function(event) {
 keys[event.key] = true
})

document.addEventListener("keyup", function(event) {
 keys[event.key] = false
})

function gameLoop() {
 if (keys["ArrowUp"] || keys["w"]) player.y -= 5
 if (keys["ArrowDown"] || keys["s"]) player.y += 5
```

```
 if (keys["ArrowLeft"] || keys["a"]) player.x -= 5
 if (keys["ArrowRight"] || keys["d"]) player.x += 5

 updatePlayer()
 requestAnimationFrame(gameLoop)
}

gameLoop()
```

Auto-complete dropdown:

```
let input = document.getElementById("search")
let dropdown = document.getElementById("suggestions")
let currentSuggestion = -1

input.addEventListener("keydown", function(event) {
 let suggestions = dropdown.querySelectorAll(".suggestion")

 if (event.key === "ArrowDown") {
 event.preventDefault()
 currentSuggestion = Math.min(currentSuggestion + 1,
suggestions.length - 1)
 highlightSuggestion(suggestions)
 }

 if (event.key === "ArrowUp") {
 event.preventDefault()
 currentSuggestion = Math.max(currentSuggestion - 1, -1)
 highlightSuggestion(suggestions)
 }

 if (event.key === "Enter" && currentSuggestion >= 0) {
 event.preventDefault()
 input.value = suggestions[currentSuggestion].textContent
 dropdown.style.display = "none"
 }

 if (event.key === "Escape") {
 dropdown.style.display = "none"
 }
```

```
})

function highlightSuggestion(suggestions) {
 suggestions.forEach((s, i) => {
 s.classList.toggle("highlighted", i === currentSuggestion)
 })
}
```

Character counter:

```
let textarea = document.getElementById("message")
let counter = document.getElementById("counter")
let maxLength = 280

textarea.addEventListener("input", function() {
 let remaining = maxLength - this.value.length
 counter.textContent = remaining + " characters remaining"

 if (remaining < 0) {
 counter.style.color = "red"
 } else if (remaining < 20) {
 counter.style.color = "orange"
 } else {
 counter.style.color = ""
 }
})
```

PRACTICE

Listen for specific key presses.

Create a keyboard shortcut (Ctrl + S).

Navigate a list with arrow keys.

Prevent certain keys in an input.

Build a simple game with keyboard controls.

WHAT YOU LEARNED

You learned keydown and keyup events.
You learned event.key gives the character.
You learned event.code gives the physical key.
You learned to check modifier keys.
You learned to create keyboard shortcuts.
You learned keyboard navigation patterns.

The next chapter teaches you about form events.

BOOK 8: EVENTS
Chapter 6: Form Events

SUBMIT EVENT

Fires when form is submitted:

```
let form = document.getElementById("login-form")

form.addEventListener("submit", function(event) {
 event.preventDefault()

 let username = form.elements.username.value
 let password = form.elements.password.value

 console.log("Submitting:", username)
})
```

Always call event.preventDefault() to stop the page from reloading.

CHANGE EVENT

Fires when value changes and element loses focus:

```
let input = document.getElementById("name")

input.addEventListener("change", function() {
 console.log("Value changed to:", this.value)
})
```

For select:

```
let select = document.getElementById("country")

select.addEventListener("change", function() {
 console.log("Selected:", this.value)
})
```

For checkbox:

```
let checkbox = document.getElementById("agree")

checkbox.addEventListener("change", function() {
 console.log("Checked:", this.checked)
})
```

For radio buttons:

```
let radios = document.querySelectorAll("input[name='color']")

radios.forEach(radio => {
 radio.addEventListener("change", function() {
 if (this.checked) {
 console.log("Selected color:", this.value)
 }
 })
})
```

INPUT EVENT

Fires immediately as value changes:

```
let input = document.getElementById("search")

input.addEventListener("input", function() {
 console.log("Current value:", this.value)
})
```

Difference from change:
- input fires on every keystroke
- change fires when element loses focus

FOCUS AND BLUR

focus - fires when element gains focus:

```
let input = document.getElementById("email")

input.addEventListener("focus", function() {
 console.log("Input focused")
 this.style.borderColor = "blue"
})
```

blur - fires when element loses focus:

```
input.addEventListener("blur", function() {
 console.log("Input blurred")
 this.style.borderColor = ""
})
```

FOCUSIN AND FOCUSOUT

Like focus and blur but they bubble:

```
let form = document.getElementById("form")

form.addEventListener("focusin", function(event) {
 console.log("Focus on:", event.target.name)
})

form.addEventListener("focusout", function(event) {
 console.log("Blur from:", event.target.name)
})
```

Use focusin/focusout for event delegation.

RESET EVENT

Fires when form is reset:

```
let form = document.getElementById("form")

form.addEventListener("reset", function(event) {
 console.log("Form reset")
})
```

Prevent reset:

```
form.addEventListener("reset", function(event) {
 let confirmed = confirm("Are you sure you want to reset?")

 if (!confirmed) {
 event.preventDefault()
 }
})
```

FORM VALIDATION

Validate on submit:

```
let form = document.getElementById("signup")

form.addEventListener("submit", function(event) {
 event.preventDefault()

 let errors = []

 let email = form.elements.email.value
 if (!email.includes("@")) {
 errors.push("Invalid email")
 }

 let password = form.elements.password.value
 if (password.length < 8) {
 errors.push("Password must be at least 8 characters")
 }

 if (errors.length > 0) {
 showErrors(errors)
 return
 }

 submitForm(form)
})
```

Validate on blur:

```
let inputs = document.querySelectorAll("input[required]")

inputs.forEach(input = > {
 input.addEventListener("blur", function() {
 validateField(this)
 })
```

```
})

function validateField(input) {
 let error = input.nextElementSibling

 if (input.value.trim() === "") {
 input.classList.add("invalid")
 error.textContent = "This field is required"
 } else {
 input.classList.remove("invalid")
 error.textContent = ""
 }
}
```

INVALID EVENT

Fires when form validation fails:

```
let input = document.getElementById("email")

input.addEventListener("invalid", function(event) {
 event.preventDefault()

 this.classList.add("invalid")
 showError(this, "Please enter a valid email")
})
```

LIVE VALIDATION

Validate as user types:

```
let emailInput = document.getElementById("email")
let emailError = document.getElementById("email-error")
```

```
emailInput.addEventListener("input", function() {
 let email = this.value

 if (email = = = "") {
 emailError.textContent = ""
 this.classList.remove("valid", "invalid")
 } else if (!isValidEmail(email)) {
 emailError.textContent = "Invalid email format"
 this.classList.add("invalid")
 this.classList.remove("valid")
 } else {
 emailError.textContent = ""
 this.classList.add("valid")
 this.classList.remove("invalid")
 }
})

function isValidEmail(email) {
 return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)
}
```

PASSWORD STRENGTH

```
let passwordInput = document.getElementById("password")
let strengthMeter = document.getElementById("strength")

passwordInput.addEventListener("input", function() {
 let password = this.value
 let strength = calculateStrength(password)

 strengthMeter.className = "strength-" + strength.level
 strengthMeter.textContent = strength.label
})

function calculateStrength(password) {
 let score = 0
```

```
 if (password.length >= 8) score++
 if (password.length >= 12) score++
 if (/[a-z]/.test(password) && /[A-Z]/.test(password)) score++
 if (/\d/.test(password)) score++
 if (/[^a-zA-Z0-9]/.test(password)) score++

 let levels = [
 { level: "weak", label: "Weak" },
 { level: "fair", label: "Fair" },
 { level: "good", label: "Good" },
 { level: "strong", label: "Strong" },
 { level: "very-strong", label: "Very Strong" }
 ]

 return levels[Math.min(score, 4)]
}
```

PRACTICAL EXAMPLES

Auto-save form:

```
let form = document.getElementById("draft")

form.addEventListener("input", debounce(function() {
 let formData = new FormData(form)
 let data = Object.fromEntries(formData)
 localStorage.setItem("draft", JSON.stringify(data))
 showSaved()
}, 1000))

function debounce(fn, delay) {
 let timeout

 return function(...args) {
 clearTimeout(timeout)
 timeout = setTimeout(() => fn.apply(this, args), delay)
 }
```

```
}

function showSaved() {
 let indicator = document.getElementById("saved")
 indicator.textContent = "Saved"
 setTimeout(() => indicator.textContent = "", 2000)
}
```

Dependent dropdowns:

```
let countrySelect = document.getElementById("country")
let citySelect = document.getElementById("city")

let cities = {
 us: ["New York", "Los Angeles", "Chicago"],
 uk: ["London", "Manchester", "Birmingham"],
 ca: ["Toronto", "Vancouver", "Montreal"]
}

countrySelect.addEventListener("change", function() {
 let country = this.value

 citySelect.innerHTML = "<option value="">Select a city</
option>"

 if (cities[country]) {
 cities[country].forEach(city => {
 let option = document.createElement("option")
 option.value = city.toLowerCase()
 option.textContent = city
 citySelect.appendChild(option)
 })
 }
})
```

Form progress indicator:

```
let form = document.getElementById("registration")
let progress = document.getElementById("progress")
let requiredFields = form.querySelectorAll("[required]")
```

```
form.addEventListener("input", updateProgress)

function updateProgress() {
 let filled = 0

 requiredFields.forEach(field => {
 if (field.value.trim() !== "") {
 filled++
 }
 })

 let percent = Math.round((filled / requiredFields.length) * 100)
 progress.style.width = percent + "%"
 progress.textContent = percent + "%"
}
```

Multi-step form:

```
let form = document.getElementById("multi-step")
let steps = form.querySelectorAll(".step")
let currentStep = 0

function showStep(index) {
 steps.forEach((step, i) => {
 step.style.display = i === index ? "block" : "none"
 })
}

form.addEventListener("click", function(event) {
 if (event.target.classList.contains("next")) {
 if (validateCurrentStep()) {
 currentStep++
 showStep(currentStep)
 }
 }

 if (event.target.classList.contains("prev")) {
 currentStep--
 showStep(currentStep)
```

```
 }
})

function validateCurrentStep() {
 let currentFields =
steps[currentStep].querySelectorAll("[required]")
 let isValid = true

 currentFields.forEach(field = > {
 if (field.value.trim() = = = "") {
 field.classList.add("invalid")
 isValid = false
 } else {
 field.classList.remove("invalid")
 }
 })

 return isValid
}

showStep(0)
```

PRACTICE

Handle form submit and prevent default.

Validate inputs on blur.

Create live validation as user types.

Build a password strength meter.

Create dependent dropdown selects.

WHAT YOU LEARNED

You learned submit event for form submission.
You learned change event for value changes.
You learned input event for live updates.
You learned focus and blur for focus tracking.
You learned validation patterns and techniques.
You learned practical form handling patterns.

The next chapter teaches you about window and document events.
BOOK 8: EVENTS
Chapter 7: Window and Document Events

PAGE LOAD EVENTS

DOMContentLoaded - fires when HTML is parsed:

```
document.addEventListener("DOMContentLoaded", function() {
 console.log("DOM is ready")
})
```

load - fires when page and all resources are loaded:

```
window.addEventListener("load", function() {
 console.log("Page fully loaded")
})
```

DOMContentLoaded fires first. It does not wait for images, stylesheets, etc.
load fires after everything including images.

Usually DOMContentLoaded is what you want.

PLACING SCRIPTS

Script at end of body:

```
<body>
 <h1>Page content</h1>
 <script src="app.js"></script>
</body>
```

The DOM is ready when the script runs. No event needed.

Script in head with defer:

```
<head>
 <script src="app.js" defer></script>
</head>
```

defer waits for DOM to be ready before running.

Script in head with DOMContentLoaded:

```
<head>
 <script>
document.addEventListener("DOMContentLoaded", function() {
// DOM is ready
})
 </script>
</head>
```

BEFORE LEAVING PAGE

beforeunload - fires before user leaves:

```
window.addEventListener("beforeunload", function(event) {
 event.preventDefault()
 event.returnValue = ""
})
```

This shows a confirmation dialog.

Use it to warn about unsaved changes:

```
let hasUnsavedChanges = false

document.getElementById("form").addEventListener("input",
function() {
 hasUnsavedChanges = true
})

document.getElementById("form").addEventListener("submit",
function() {
 hasUnsavedChanges = false
})

window.addEventListener("beforeunload", function(event) {
 if (hasUnsavedChanges) {
 event.preventDefault()
 event.returnValue = ""
 }
})
```

UNLOAD EVENT

unload - fires when page is being unloaded:

```
window.addEventListener("unload", function() {
 console.log("Page is unloading")
})
```

Limited use. You cannot show dialogs or prevent unload here.

RESIZE EVENT

Fires when window size changes:

```
window.addEventListener("resize", function() {
 console.log("Width:", window.innerWidth)
 console.log("Height:", window.innerHeight)
})
```

Debounce for performance:

```
let resizeTimeout

window.addEventListener("resize", function() {
 clearTimeout(resizeTimeout)

 resizeTimeout = setTimeout(function() {
 console.log("Resize finished")
 handleResize()
 }, 250)
})

function handleResize() {
 if (window.innerWidth < 768) {
 console.log("Mobile view")
 } else {
 console.log("Desktop view")
 }
}
```

SCROLL EVENT

Fires when page is scrolled:

```
window.addEventListener("scroll", function() {
 console.log("Scroll position:", window.scrollY)
})
```

Throttle for performance:

```
let isScrolling = false

window.addEventListener("scroll", function() {
 if (!isScrolling) {
 isScrolling = true

 requestAnimationFrame(function() {
 handleScroll()
 isScrolling = false
 })
 }
})

function handleScroll() {
 let scrollTop = window.scrollY

 if (scrollTop > 100) {
 document.body.classList.add("scrolled")
 } else {
 document.body.classList.remove("scrolled")
 }
}
```

SCROLL POSITION

Get current scroll position:

```
let scrollTop = window.scrollY
let scrollLeft = window.scrollX

let scrollTop = document.documentElement.scrollTop
let scrollLeft = document.documentElement.scrollLeft
```

Scroll to position:

```
window.scrollTo(0, 500)

window.scrollTo({
 top: 500,
 behavior: "smooth"
})
```

Scroll by amount:

```
window.scrollBy(0, 100)

window.scrollBy({
 top: 100,
 behavior: "smooth"
})
```

VISIBILITY CHANGE

Fires when tab becomes visible or hidden:

```
document.addEventListener("visibilitychange", function() {
 if (document.hidden) {
 console.log("Tab is hidden")
 } else {
 console.log("Tab is visible")
 }
})
```

Pause video when tab hidden:

```
let video = document.getElementById("video")

document.addEventListener("visibilitychange", function() {
 if (document.hidden) {
 video.pause()
 } else {
```

```
 video.play()
 }
})
```

ONLINE AND OFFLINE

Detect network status:

```
window.addEventListener("online", function() {
 console.log("Back online")
 showNotification("You are back online")
})

window.addEventListener("offline", function() {
 console.log("Gone offline")
 showNotification("You are offline")
})
```

Check current status:

```
if (navigator.onLine) {
 console.log("Currently online")
} else {
 console.log("Currently offline")
}
```

HASH CHANGE

Fires when URL hash changes:

```
window.addEventListener("hashchange", function() {
 console.log("Hash changed to:", window.location.hash)
})
```

Simple routing:

```
window.addEventListener("hashchange", handleRoute)

function handleRoute() {
 let hash = window.location.hash.slice(1)

 document.querySelectorAll(".page").forEach(page => {
 page.style.display = "none"
 })

 let page = document.getElementById(hash) ||
document.getElementById("home")
 page.style.display = "block"
}

handleRoute()
```

POPSTATE

Fires when browser back/forward buttons used:

```
window.addEventListener("popstate", function(event) {
 console.log("Navigation:", event.state)
})
```

With history API:

```
function navigate(path, data) {
 history.pushState(data, "", path)
 loadPage(path)
}

window.addEventListener("popstate", function(event) {
 loadPage(window.location.pathname)
})
```

PRACTICAL EXAMPLES

Sticky header:

```
let header = document.getElementById("header")
let headerOffset = header.offsetTop

window.addEventListener("scroll", function() {
 if (window.scrollY > headerOffset) {
 header.classList.add("sticky")
 } else {
 header.classList.remove("sticky")
 }
})
```

Back to top button:

```
let backToTop = document.getElementById("back-to-top")

window.addEventListener("scroll", function() {
 if (window.scrollY > 300) {
 backToTop.style.display = "block"
 } else {
 backToTop.style.display = "none"
 }
})

backToTop.addEventListener("click", function() {
 window.scrollTo({
 top: 0,
 behavior: "smooth"
 })
})
```

Reading progress:

```
let progress = document.getElementById("reading-progress")
```

```
window.addEventListener("scroll", function() {
 let scrollTop = window.scrollY
 let docHeight = document.documentElement.scrollHeight -
window.innerHeight
 let percent = (scrollTop / docHeight) * 100

 progress.style.width = percent + "%"
})
```

Lazy loading images:

```
function lazyLoad() {
 let images = document.querySelectorAll("img[data-src]")

 images.forEach(img = > {
 let rect = img.getBoundingClientRect()

 if (rect.top < window.innerHeight && rect.bottom > 0) {
 img.src = img.dataset.src
 img.removeAttribute("data-src")
 }
 })
}

window.addEventListener("scroll", lazyLoad)
window.addEventListener("load", lazyLoad)
```

Responsive navigation:

```
let nav = document.getElementById("nav")
let menuToggle = document.getElementById("menu-toggle")

menuToggle.addEventListener("click", function() {
 nav.classList.toggle("open")
})

window.addEventListener("resize", function() {
 if (window.innerWidth > 768) {
 nav.classList.remove("open")
```

```
 }
})
```

PRACTICE

Handle DOMContentLoaded event.

Detect scroll position and show/hide elements.

Handle window resize events.

Create a sticky header.

Build a reading progress indicator.

WHAT YOU LEARNED

You learned DOMContentLoaded for DOM ready.
You learned load for full page load.
You learned beforeunload for unsaved changes warning.
You learned resize and scroll events.
You learned visibilitychange for tab focus.
You learned online/offline for network status.
You learned hashchange and popstate for routing.

The next chapter brings everything together with practice programs.

KEYBOARD PIANO

Play notes with keyboard:

HTML:

```
<div id="piano">
 <div class="key white" data-note="C" data-key="a">A</div>
 <div class="key black" data-note="C#" data-key="w">W</
div>
 <div class="key white" data-note="D" data-key="s">S</div>
 <div class="key black" data-note="D#" data-key="e">E</div>
 <div class="key white" data-note="E" data-key="d">D</div>
 <div class="key white" data-note="F" data-key="f">F</div>
 <div class="key black" data-note="F#" data-key="t">T</div>
 <div class="key white" data-note="G" data-key="g">G</div>
</div>
```

JavaScript:

```
let piano = document.getElementById("piano")

let audioContext = new (window.AudioContext ||
window.webkitAudioContext)()

let frequencies = {
 "C": 261.63,
 "C#": 277.18,
 "D": 293.66,
 "D#": 311.13,
 "E": 329.63,
 "F": 349.23,
 "F#": 369.99,
 "G": 392.00
}

function playNote(note) {
 let oscillator = audioContext.createOscillator()
 let gainNode = audioContext.createGain()
```

```
 oscillator.connect(gainNode)
 gainNode.connect(audioContext.destination)

 oscillator.frequency.value = frequencies[note]
 oscillator.type = "sine"

 gainNode.gain.setValueAtTime(0.5, audioContext.currentTime)
 gainNode.gain.exponentialRampToValueAtTime(0.01,
audioContext.currentTime + 1)

 oscillator.start(audioContext.currentTime)
 oscillator.stop(audioContext.currentTime + 1)
}

piano.addEventListener("click", function(event) {
 let key = event.target.closest(".key")

 if (key) {
 let note = key.dataset.note
 playNote(note)
 key.classList.add("active")
 setTimeout(() => key.classList.remove("active"), 100)
 }
})

document.addEventListener("keydown", function(event) {
 if (event.repeat) return

 let key = piano.querySelector(`[data-
key="${event.key.toLowerCase()}"]`)

 if (key) {
 let note = key.dataset.note
 playNote(note)
 key.classList.add("active")
 }
})

document.addEventListener("keyup", function(event) {
 let key = piano.querySelector(`[data-
```

```
key="${event.key.toLowerCase()}"]`)

 if (key) {
 key.classList.remove("active")
 }
})
```

WHACK-A-MOLE GAME

Click moles as they appear:

HTML:

```
<div id="game">
 <div id="score">Score: <span>0</span></div>
 <div id="time">Time: <span>30</span>s</div>
 <div id="grid">
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 <div class="hole"></div>
 </div>
 <button id="start">Start Game</button>
</div>
```

JavaScript:

```
let grid = document.getElementById("grid")
let scoreDisplay = document.querySelector("#score span")
let timeDisplay = document.querySelector("#time span")
let startButton = document.getElementById("start")
```

```
let score = 0
let timeLeft = 30
let gameInterval = null
let moleInterval = null

function showMole() {
 let holes = grid.querySelectorAll(".hole")
 let activeHole = grid.querySelector(".hole.active")

 if (activeHole) {
 activeHole.classList.remove("active")
 }

 let randomIndex = Math.floor(Math.random() * holes.length)
 holes[randomIndex].classList.add("active")

 setTimeout(() => {
 holes[randomIndex].classList.remove("active")
 }, 800)
}

function startGame() {
 score = 0
 timeLeft = 30
 scoreDisplay.textContent = score
 timeDisplay.textContent = timeLeft
 startButton.disabled = true

 moleInterval = setInterval(showMole, 1000)

 gameInterval = setInterval(() => {
 timeLeft--
 timeDisplay.textContent = timeLeft

 if (timeLeft === 0) {
 endGame()
 }
 }, 1000)
}
```

```
function endGame() {
 clearInterval(gameInterval)
 clearInterval(moleInterval)

 let activeHole = grid.querySelector(".hole.active")
 if (activeHole) {
 activeHole.classList.remove("active")
 }

 startButton.disabled = false
 alert(`Game Over! Final Score: ${score}`)
}

grid.addEventListener("click", function(event) {
 let hole = event.target.closest(".hole")

 if (hole && hole.classList.contains("active")) {
 score++
 scoreDisplay.textContent = score
 hole.classList.remove("active")
 hole.classList.add("hit")
 setTimeout(() => hole.classList.remove("hit"), 200)
 }
})

startButton.addEventListener("click", startGame)
```

SHOPPING CART

Add and remove items:

HTML:

```
<div id="shop">
 <div class="products">
 <div class="product" data-id="1" data-name="Laptop" data-
price="999">
```

```
  <h3>Laptop</h3>
  <p>$999</p>
  <button class="add-to-cart">Add to Cart</button>
  </div>
  <div class="product" data-id="2" data-name="Mouse" data-
price="29">
  <h3>Mouse</h3>
  <p>$29</p>
  <button class="add-to-cart">Add to Cart</button>
  </div>
  <div class="product" data-id="3" data-name="Keyboard" data-
price="79">
  <h3>Keyboard</h3>
  <p>$79</p>
  <button class="add-to-cart">Add to Cart</button>
  </div>
  </div>

  <div id="cart">
  <h2>Shopping Cart</h2>
  <ul id="cart-items"></ul>
  <p id="total">Total: $0</p>
  </div>
</div>
```

JavaScript:

```
let cart = []
let cartItems = document.getElementById("cart-items")
let totalDisplay = document.getElementById("total")

function updateCart() {
 cartItems.innerHTML = ""

 cart.forEach((item, index) => {
 let li = document.createElement("li")
 li.innerHTML = `
 ${item.name} x ${item.quantity} - $${item.price * item.quantity}
 <button class="increase" data-index="${index}">+</button>
 <button class="decrease" data-index="${index}">-</button>
```

```
 <button class="remove" data-index="${index}">Remove</
button>
 `
 cartItems.appendChild(li)
 })

 let total = cart.reduce((sum, item) => sum + item.price *
item.quantity, 0)
 totalDisplay.textContent = `Total: $${total}`
}

document.querySelector(".products").addEventListener("click",
function(event) {
 if (event.target.classList.contains("add-to-cart")) {
 let product = event.target.closest(".product")
 let id = product.dataset.id
 let name = product.dataset.name
 let price = parseInt(product.dataset.price)

 let existing = cart.find(item => item.id === id)

 if (existing) {
 existing.quantity++
 } else {
 cart.push({ id, name, price, quantity: 1 })
 }

 updateCart()
 }
})

cartItems.addEventListener("click", function(event) {
 let index = event.target.dataset.index

 if (index === undefined) return

 index = parseInt(index)

 if (event.target.classList.contains("increase")) {
 cart[index].quantity++
```

```
 }

 if (event.target.classList.contains("decrease")) {
 cart[index].quantity--

 if (cart[index].quantity === 0) {
 cart.splice(index, 1)
 }
 }

 if (event.target.classList.contains("remove")) {
 cart.splice(index, 1)
 }

 updateCart()
})
```

TYPING SPEED TEST

Test typing speed:

HTML:

```
<div id="typing-test">
 <p id="sample-text">The quick brown fox jumps over the lazy
dog.</p>
 <input type="text" id="input" placeholder="Start typing..."
disabled>
 <div id="stats">
 <span>Time: <span id="time">0</span>s</span>
 <span>WPM: <span id="wpm">0</span> </span>
 <span>Accuracy: <span id="accuracy">100</span>%</
span>
 </div>
 <button id="start-test">Start Test</button>
</div>
```

JavaScript:

```
let sampleText = document.getElementById("sample-text")
let input = document.getElementById("input")
let timeDisplay = document.getElementById("time")
let wpmDisplay = document.getElementById("wpm")
let accuracyDisplay = document.getElementById("accuracy")
let startButton = document.getElementById("start-test")

let startTime = null
let timerInterval = null
let originalText = sampleText.textContent

function startTest() {
 input.value = ""
 input.disabled = false
 input.focus()
 startTime = null
 timeDisplay.textContent = "0"
 wpmDisplay.textContent = "0"
 accuracyDisplay.textContent = "100"
 startButton.disabled = true
 sampleText.innerHTML = originalText
}

function calculateStats() {
 let typed = input.value
 let original = originalText.substring(0, typed.length)

 let correctChars = 0
 for (let i = 0; i < typed.length; i++) {
 if (typed[i] === original[i]) {
 correctChars++
 }
 }

 let accuracy = typed.length > 0 ? Math.round((correctChars /
typed.length) * 100) : 100
 accuracyDisplay.textContent = accuracy
```

```
 if (startTime) {
 let elapsed = (Date.now() - startTime) / 1000 / 60
 let words = typed.length / 5
 let wpm = Math.round(words / elapsed)
 wpmDisplay.textContent = wpm
 }
}

function highlightText() {
 let typed = input.value
 let html = ""

 for (let i = 0; i < originalText.length; i++) {
 if (i < typed.length) {
 if (typed[i] === originalText[i]) {
 html += `<span class="correct">${originalText[i]}</span>`
 } else {
 html += `<span class="incorrect">${originalText[i]}</span>`
 }
 } else {
 html += originalText[i]
 }
 }

 sampleText.innerHTML = html
}

input.addEventListener("input", function() {
 if (!startTime) {
 startTime = Date.now()

 timerInterval = setInterval(() => {
 let elapsed = Math.floor((Date.now() - startTime) / 1000)
 timeDisplay.textContent = elapsed
 }, 100)
 }

 highlightText()
 calculateStats()
```

```
 if (input.value === originalText) {
 clearInterval(timerInterval)
 input.disabled = true
 startButton.disabled = false
 alert(`Completed! WPM: ${wpmDisplay.textContent}, Accuracy:
${accuracyDisplay.textContent}%`)
 }
})

startButton.addEventListener("click", startTest)
```

COLOR PICKER

Interactive color selector:

HTML:

```
<div id="color-picker">
 <div id="preview" style="background: #ff0000;"></div>
 <div class="sliders">
 <label>Red: <input type="range" id="red" min="0"
max="255" value="255"></label>
 <label>Green: <input type="range" id="green" min="0"
max="255" value="0"></label>
 <label>Blue: <input type="range" id="blue" min="0"
max="255" value="0"></label>
 </div>
 <input type="text" id="hex" value="#ff0000" readonly>
 <button id="copy">Copy Hex</button>
</div>
```

JavaScript:

```
let preview = document.getElementById("preview")
let redSlider = document.getElementById("red")
let greenSlider = document.getElementById("green")
let blueSlider = document.getElementById("blue")
```

```
let hexInput = document.getElementById("hex")
let copyButton = document.getElementById("copy")

function updateColor() {
 let r = parseInt(redSlider.value)
 let g = parseInt(greenSlider.value)
 let b = parseInt(blueSlider.value)

 let hex = "#" + [r, g, b].map(x => x.toString(16).padStart(2,
"0")).join("")

 preview.style.background = hex
 hexInput.value = hex
}

redSlider.addEventListener("input", updateColor)
greenSlider.addEventListener("input", updateColor)
blueSlider.addEventListener("input", updateColor)

copyButton.addEventListener("click", function() {
 navigator.clipboard.writeText(hexInput.value).then(() => {
 let originalText = copyButton.textContent
 copyButton.textContent = "Copied!"
 setTimeout(() => copyButton.textContent = originalText, 1500)
 })
})
```

WHAT YOU BUILT

You built five complete programs:

Keyboard piano with mouse and keyboard controls
Whack-a-mole game with scoring and timer
Shopping cart with add, remove, and update
Typing speed test with WPM and accuracy
Color picker with sliders and copy

Each program demonstrated:

Event delegation for dynamic elements
Multiple event types (click, keydown, input)
State management and updates
User interaction patterns
Real-time feedback

WHAT COMES NEXT

You now understand JavaScript events:

- Event basics and the event object (Book 8, Chapter 1)
- Event flow and propagation (Book 8, Chapter 2)
- Event delegation (Book 8, Chapter 3)
- Mouse events (Book 8, Chapter 4)
- Keyboard events (Book 8, Chapter 5)
- Form events (Book 8, Chapter 6)
- Window and document events (Book 8, Chapter 7)

Book 9 teaches you asynchronous JavaScript. You will:

- Understand callbacks
- Work with Promises
- Use async/await
- Handle timing with setTimeout
- Fetch data from APIs

Events are synchronous triggers. Async code handles things that take time.

You finished Book 8. You can make pages interactive. Now we handle time and waiting.